Building Wire Cell Software (prototype and toolkit)

Brett Viren

Physics Department

BROOKHAVEN
NATIONAL LABORATORY

Wire Cell Seminar 2015 Dec 7-9

Outline

Major External Dependencies

Wire Cell prototype and toolkit themselves are fairly easy to build and install but rely on a few "heavy" prerequisites.

package	prototype	toolkit	notes
C++11 compiler	required	required	GCC 4.9.2/Clang 7
ROOT6	required (core)	**optional** (tests + I/O pkgs)	get latest
BOOST	required	required	$\geq$ 1.55
Python	optional/required (dictionaries and PyROOT)		2.7.x need for `waf`
Eigen3	required	soon	header-only pkg
JSONCPP	✗	required	
Intel TBB	✗	required	
FFT imp	ROOT/fftw3	TBD	

Especially for "core" toolkit we try to limit external dependencies. ROOT is a problem, BOOST may be a problem.

Automated Build of Externals

Wire Cell Prototype provides Worch-based build automation for the externals.

- Maybe hasn't kept up.
- There are so few external packages that it may be more hassle than it's worth.

For details, see:

`https://github.com/BNLIF/wire-cell-externals`
`https://github.com/WireCell/wire-cell-externals`

Compiler

- Wire Cell uses C++11 and maybe some C++14 sneaking in.
- It's up to you to provide a working compiler.
- Most but not all recent OSes do:
  - Debian world has had GCC 4.9.2+ for a while.
  - Recent Mac has Clang 7.
  - RHEL (ie Sci. Linux) 7 has GCC 4.8, native - untested.
  - `fnal.gov` has GCC 4.9.2 as UPS product

ROOT6 - from source

Configure for C++11, FFTW3, Minuit and Python.

```
$ wget https://root.cern.ch/download/root_v6.05.02.source.tar.gz
$ tar -xvf root_v6.05.02.source.tar.gz
$ mkdir root-build
$ cd root-build/
$ cmake -DCMAKE_INSTALL_PREFIX={install_dir} \
-Dbuiltin_xrootd=ON -Dcxx11=ON -Dpythia6=OFF -Dminuit2=ON \
-Dpython=ON \
-DFFTW_INCLUDE_DIR=/usr/include \
-DFFTW_LIBRARY=/usr/lib/x86_64-linux-gnu/libfftw3.so \
../root/
$ make
$ make install
```

For this manual installation, activate ROOT environment using the (ROOT) standard:

```
$ source /path/to/install/bin/thisroot.sh
```

Debian and co.

Recent Debian (Ubuntu, etc) can provide everything except ROOT6:

```
$ sudo apt-get install \
    git build-essential gcc g++ make cmake \
    python-dev libboost-all-dev \
    libsqlite3-dev tcl8.5-dev \
    libxpm-dev libxft-dev libXext-dev \
    libeigen3-dev
```

Maybe others are needed...

Mac OS X + Homebrew

On Mac (tested on 10.10.5, "Yosemite") you get everything for cheap with homebrew!

```
$ brew install python
$ brew install boost --with-python
$ brew install homebrew/science/root6
```

If you are new to homebrew see `http://brew.sh/`.
The native Mac compiler is now Clang but with the command confusingly still accessible as "`g++`".

Fermilab UPS

This worked at one time to provide ROOT and BOOST:

```
$ source /grid/fermiapp/products/larsoft/setups
$ setup root v6_04_02 -q e7:prof
$ setup boost v1_57_0 -q e7:prof
```

This also puts a suitable `g++` in your `$PATH`.

BNL RACF

A build of ROOT 6 is available on RACF:

```
$ source /gpfs01/lbne/users/sw/wc/bin/thisroot.sh
```

This will also set up to use `g++` 4.9.2.

Prototype Software Repositories

All wire cell **prototype** source is in the "**BNLIF**" GitHub organization:

`https://github.com/BNLIF/`

Git submodules are used to aggregate each package from this top level one:

`https://github.com/BNLIF/wire-cell`

By default, git submodules are hooked in assuming SSH access (not HTTPS).

Cloning the source

Developers do:

```
$ git clone git@github.com:BNLIF/wire-cell.git
$ cd wire-cell/
```

Others may anonymously do:

```
$ git clone https://github.com/BNLIF/wire-cell.git
$ cd wire-cell/
$ ./switch-git-urls
```

Everyone does:

```
$ git submodule init
$ git submodule update
```

Building Wire Cell Prototype software

Wire Cell Prototype software is built with **plain waf** using the provided `waf-tools` package:

```
$ alias waf=`pwd`/waf-tools/waf
$ waf --prefix=/path/to/install configure build install
```

That's it.

Build options

See available options

```
$ waf --help
```

Configure to build with debug symbols:

```
$ waf --build-debug=-ggdb3 [...others...] configure
$ waf clean build install
```

Build Doxygen docs:

```
$ waf --doxygen-tarball=doxy.tar
```

Run-time Environment

Externals must be setup in whatever manner is associated with how they were built.
For Wire Cell itself one needs to set the usual:

- `PATH`
- `LD_LIBRARY_PATH`

to point to appropriate directories under `/path/to/install`.

Example Job (prototype)

```
$ wire-cell-improve \
   ChannelWireGeometry_v2.txt celltree-pi0-new.root  0
$ wire-cell-mc-ns1 \
   ChannelWireGeometry_v2.txt shower3D_cluster_0.root 0
$ cd bee/  # for now must be in wire-cell/bee/ source dir
$ python dump_json.py ../shower3D_cluster_0.root simple charge truth
$ ./upload-to-bee.sh to_upload.zip
```

Or manually (click-drag) upload `to_upload.zip` file to `http://www.phy.bnl.gov/wire-cell/bee/`.

File links:

- ChannelWireGeometry_v2.txt
- celltree-pi0-new.root
- more celltree files

Note: can give URL instead of local ROOT file.

```
$ wire-cell-improve ChannelWireGeometry_v2.txt \
 http://lycastus.phy.bnl.gov/~wire-cell/examples/celltree/3.0/celltree-pi0-new.root 0
```

Wire Cell Toolkit Repositories

All wire cell **toolkit** source is in the "**WireCell**" GitHub organization:

https://github.com/WireCell/

Git submodules are used to aggregate each package from this top level one:

https://github.com/WireCell/wire-cell-build

Same guidance as for prototype:

```
$ git clone git@github.com:WireCell/wire-cell-build.git
$ cd wire-cell-build/
$ ./switch-git-urls  # <-- if not a developer
$ git submodule init
$ git submodule update
```

Building Wire Cell Toolkit software

Wire Cell Toolkit software is built with **waf** using the provided **waf-bundled** command `wcb` (no `waf-tools` package needed).

```
$ ./wcb --prefix=/path/to/install configure build install
```

If your dependencies are not found by `wcb` then:

```
$ ./wcb --help
$ ./wcb --boost-libs=... --boost-includes=... \
        --with-root=... configure
$ ./wcb  # <-- default command is "build"
$ ./wcb install
```

Run-time Environment

Externals must be setup in whatever manner is associated with how they were built.
For Wire Cell itself one needs to set the usual:

- `PATH`
- `LD_LIBRARY_PATH`

to point to appropriate directories under `/path/to/install`.

Example jobs (toolkit)

All unit tests are run during build (check terminal output) and can be run explicitly by hand.

```
$ ls build/*/test_*
```

Some tests generate PDF files and some will alternatively give a live `TCanvas` if a command line argument is passed. Eg:

```
$ ./build/gen/test_wireparams foo
$ ./build/gen/test_drifter foo
$ ./build/gen/test_boundcells
$ ./build/gen/test_boundcells_dune
$ ./build/gen/test_diffuser
$ ./build/alg/test_depodriftcell_sync
```

Eventually:

```
$ wire-cell -c myjob.cfg -p myplugin.so -o myoutput.root myinput.root
```

Current full-chain test

Toolkit full-chain using TBB `data_flow`. Default concurrency number equals #CPUs.

```
$ ./build/tbb/test_tbb_dfp [concurrency]
...
DepoSource: 0/660
Drifter U:660/660 V:660/660 W:660/660
Diffuser U:660/660 V:660/660 W:660/660
Ductor U:660/83 V:660/82 W:660/83
Merger:246/82
Digitizer:82/82
CellSelector:82/82
```

Shown: counts of input/output objects to some of the DFP nodes. Can you spot the bug?

Start hacking!